Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

QoS-aware placement of interdependent services in energy-harvesting-enabled multi-access edge computing

Shuyi Chen [a,b], Panagiotis Oikonomou [c], Zhengchang Hua [a,b], Nikos Tziritas [c], Karim Djemame [b], Nan Zhang [a], Georgios Theodoropoulos [a,*]

[a] Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, 518000, Guangdong, China
[b] School of Computer Science, University of Leeds, Leeds, LS2 9JT, West Yorkshire, UK
[c] Department of Informatics and Telecommunications, University of Thessaly, Lamia, 35100, Greece

ARTICLE INFO

Keywords:
Service placement
Multi-access edge computing
Energy harvesting
Task dependency graph

ABSTRACT

The advent of 5G drives the growth of multi-access edge computing (MEC), a revolutionary paradigm that utilises edge resources to enable low-latency mobile access and support complex service execution. Deploying services across geographically distributed edge nodes challenges providers to optimise performance metrics like end-to-end latency and resource efficiency, impacting user experience, operational cost, and environmental footprint. The energy harvesting (EH) technology provides clean and renewable energy at the edge, promoting the MEC system to minimise the impacts on the environment. However, the integration of EH can introduce energy limits and uncertainty to the powered devices. In the context of service scheduling with data flow dependencies, we propose two offline and heuristic-based service placement algorithms that balance minimising latency and maximising resource efficiency with fast execution. The two algorithms, evaluated in a simulated environment using state-of-the-art workload benchmarks, achieve significant energy consumption improvements while maintaining comparable latency. Based on the designed algorithms, we take a step further by developing an online dynamic resource scheduling and service offloading approach for MEC systems with EH capabilities. Simulation results demonstrate that the proposed strategy effectively utilise the harvested energy while granting a low user-experienced latency and low operational cost.

1. Introduction

Driven by the rise of fifth generation (5G) network, Multi-access Edge Computing (MEC), a network architecture that integrates computing and storage capabilities into Micro Data Centers (MDCs) located near base stations, emerges as a transformative paradigm. Leveraging network and computing resources at the network edge offers low-latency access for mobile subscribers while facilitating the execution of complex, computationally intensive applications [1]. However, effectively orchestrating custom applications in MEC requires strategic allocation of resources. In the context of resource scheduling, this involves managing computation and network resources distributed across heterogeneous edge nodes to efficiently serve user requests originating nearby, and balancing the demands of both users (fast response) and providers (resource efficiency, cost, etc.)

User-submitted jobs may encompass IoT data processing, healthcare, Augmented Reality-based experiences, and financial services. Platform providers must allocate adequate edge resources to ensure the successful execution of applications while balancing the demands of both users and providers.

Multi-access edge computing (MEC) operates in a dynamic environment characterised by high user capacity and fluctuating user location. This necessitates adaptive task coordination to handle requests from numerous mobile subscribers as their positions change. Compared to centralised cloud datacenters, MEC faces limitations in communication, computation, and storage capacity due to the smaller physical footprint of edge nodes. Additionally, edge devices within the resource pool may have limited energy capacity and power limits. Furthermore, managing geographically dispersed edge devices requires constant monitoring of their status changes, including outages, network fluctuations, and even user location changes. Therefore, effectively allocating limited resources and ensuring a stable user experience within this dynamic edge environment presents a significant challenge for MEC systems.

With the growing trend of low-carbon edge computing, to use the resources in a more sustainable and environmental-friendly way, the energy harvesting (EH) technology can be considered an approach

* Corresponding author.
E-mail address: theogeorgios@gmail.com (G. Theodoropoulos).

https://doi.org/10.1016/j.future.2025.108009
Received 21 January 2025; Received in revised form 29 April 2025; Accepted 30 June 2025
Available online 11 July 2025

to harness renewable energy from sources such as solar, wind, and suitable energy sources. MEC systems equipped with EH devices is able to power the computing devices with clean energy and enable green communications [2]. Moreover, EH devices can harvest energy from ambient sources to power MEC systems in remote regions, reducing reliance on traditional power infrastructure and promoting sustainable edge computing. However, the energy generation of such sources may be stochastically affected by environmental conditions like weather and season, and the total available energy is also limited by the device's battery storage capacity. Therefore, it is crucial to coordinate task scheduling and energy management to maximise system performance.

Quality of Service (QoS), a crucial measure of service effectiveness, is paramount for service providers in MEC. Delivering optimal QoS involves meeting multiple objectives, such as minimising latency and maximising availability. Providers achieve this through meticulous coordination of network and computing resources, including the allocation of individual tasks [3]. Beyond QoS, profitability remains a key concern. Minimising server rental costs and power consumption directly impact profits and contribute to a more sustainable industry. However, achieving these goals often involves trade-offs [4]. Balancing user experience and provider profits presents a complex multi-objective optimisation problem. While existing research offers solutions for various scenarios, there is a need for more universal and flexible approaches to address the dynamic nature of MEC environments and the complexity of modern applications.

Beyond managing the distributed nature of MEC resources, the placement of applications presents an additional challenge. In real-world applications, numerous interdependent components frequently collaborate [5]. Those underlying components, often referred to as services, each executes a specific task such as data extraction, transformation, loading, or integration. The optimal placement of these services has to fulfil the requirements of each one with dependency guarantees. The Distributed Dataflow (DDF) paradigm offers a well-suited approach for structuring these applications, as it provides a clear representation of data flow and processing steps. It utilises a Directed Acyclic Graph (DAG) to represent the flow of data and processing steps within an application, so that inter-service dependencies and the order of execution can be depicted.

Existing service placement strategies in edge computing struggle to effectively handle these complex, dependency-aware applications. Traditional methods established the importance of properly allocating computing resources to applications, but often overlook the interdependencies of application modules, or leverage a specific architecture or model, thus may falter when task dependencies are present, rendering them unsuitable for such scenarios. To address this gap, we propose service placement approaches specifically designed for Cloud-MEC environments, aiming at efficiently allocate resources for complex service execution and fulfil both user and provider demands. Offloading occurs at edge nodes located close to user devices, where computational tasks are transferred to reduce latency. Computation-intensive tasks may also be offloaded to powerful cloud datacenters, ultimately enabling the fulfilment of customised user requirements. Since finding the provably optimal resource schedule in a dynamic MEC environment is computationally intractable (NP-hard) [3], we employ heuristic-based algorithms, aiming to find high-quality, practical solutions within a reasonable time frame by using efficient rules or approximations, rather than guaranteeing optimality.

This work addresses the service placement problem with precedence constraints among service applications. Our objective is to improve quality of service (QoS), specifically minimising latency, and optimise resource efficiency, measured by energy consumption and operational cost. The inherent complexity of considering these multiple factors motivates our development of two novel heuristic-based placement algorithms and an extended online resource scheduling strategy for dependent services. These algorithms strive to achieve a balance between different optimisation goals. Our contributions in this work are threefold:

- Dependency-aware Service Placement Algorithms: We propose two novel service placement algorithms specifically designed for the multi-access edge computing (MEC) environment. These algorithms consider precedence constraints between service components to optimise both end-to-end latency and dynamic energy consumption.
- Online Dynamic Service Offloading and Resource Scheduling Algorithm: Building upon the two algorithms and incorporating the EH technology, we proposed an online dynamic service offloading and resource scheduling strategy for EH-enabled MEC system. Our approach takes the energy limitation and deadline constraints into account, maximising the utilisation of renewable energy while granting the end-to-end latency.
- YAFS Platform Extension: To support the evaluation of our algorithms, we developed an extension[1] to the YAFS simulation platform [6]. This extension enables the modelling of sequential task processing, a crucial aspect of service execution in MEC.

The remainder of this paper is organised as follows. Section 2 briefly summarises existing approaches and identifies the research gap. Section 3 describes the system models and problem formulation. Section 4 proposes our two offline algorithms in detail. Section 5 introduced our online scheduling strategies based on the two placement algorithms. Simulation results are presented in Section 6, and the paper concludes in Section 7.

2. Related work

The multi-objective placement problem has attracted significant research attention, with solutions targeting diverse deployment scenarios and optimisation goals. Thorough survey of the literature on placement problems across various computing paradigms can be found in [3].

Researchers have developed a variety of resource management approaches for application placement by integrating specific platforms and architectures. For example, [7] solves the joint user association and service function chain (SFC) placement problem in 5G networks, [8] proposed Kubernetes-based container resource management schemes for Industrial IoT applications in Cloud-Edge networks. [9] introduced an architectural approach for placing services onto cluster nodes that offer lower end-to-end latency. [10] designed a resource management scheme for Industrial IoT applications in Edge-Cloud networks, and [11] proposed a container-base scheduling strategy in Cloud-IoT environment to find suitable containers for task processing. However, leveraging specific architecture limits the applicability of those approaches. Consequently, their direct application or adaptation to our context is not feasible.

Existing task placement research often focuses on coarse-grained abstractions of applications, overlooking task dependencies. While this approach may suffice for workloads with less stringent latency requirements, several methods adopt fine-grained service placement strategies that consider inter-service dependencies, enabling more precise control for latency-sensitive applications. For instance, [12] leverages analysis tools to extract function-level dependencies from applications, aiming to minimise overall service delay. Similarly, [13] explores the offloading of dependent sub-tasks and communication resource allocation in unmanned aerial vehicle-assisted MEC systems, targeting a reduction in average user latency. However, the above approaches primarily focus on optimising a single metric or incorporate specific techniques such as service replication. In contrast, our work seeks to minimise both total energy consumption and latency during application execution, without relying on service replication.

Service placement solutions typically target various objectives, including minimising end-to-end latency [7,16], reducing costs [5], lowering energy consumption [22], and improving resource utilisation

[1] https://github.com/Sukiiichan/YAFS_MEC.

Table 1
Comparison of related placement algorithms in the literature.

Work	Platform	Algorithm	Dependency	Latency	Energy	Cost	Throughput
[14]	Fog	Genetic		✓		✓	
[15]	Fog-Cloud	Meta-heuristic		✓	✓		
[4]	MEC-EH	Lyapunov optimisation		✓	✓		
[16]	Edge	Heuristic	✓	✓			
[17]	Edge	Meta-heuristic	✓	✓			✓
[7]	5G networks	Heuristic	✓	✓			
[18]	MEC	Learning-based		✓		✓	
[19]	MEC	Heuristic	✓	✓	✓		
[20]	MEC-EH	Lyapunov optimisation		✓	✓	✓	
[21]	MEC-EH	Lyapunov optimisation		✓	✓	✓	
[22]	Fog-Cloud	Heuristic	✓	✓	✓		
[5]	MEC	Heuristic	✓	✓	✓		
[12]	MEC	Heuristic	✓	✓			
[13]	Fog/Edge	Heuristic	✓	✓	✓		
[23]	Edge-EH	Meta-heuristic				✓	
[24]	Edge-EH	Lyapunov optimisation		✓	✓		
[25]	Edge-EH	Learning-based		✓			
[26]	MEC-EH	Meta-heuristic		✓		✓	
This work	MEC-EH	Heuristic	✓	✓	✓	✓	

[14]. To provide a clear and intuitive comparison, Table 1 summarises the differences between our proposed approach and existing service placement strategies in edge computing systems. In the context of multi-objective optimisation problems, researchers have explored a wide range of methodologies, including heuristics, meta-heuristics, and deep reinforcement learning, to achieve a balance between different objectives. For example, [18] proposes a joint placement algorithm for non-scalable services, balancing latency and deployment cost. Similarly, work in [17] target joint optimisation of throughput, latency, and deployment cost for parallelisable stream processing tasks based on meta-heuristics. Integrating with edge intelligence, work in [25] consider the joint service placement and request scheduling problem optimising the intelligence model accuracy and the service delay at the same time. The two placement algorithms we propose aim at optimising both user experience and energy efficiency. We strive to minimise the end-to-end latency and reduce the dynamic energy consumption at the server caused by computations.

Several prior studies have explored optimisation objectives that converge with those targeted by our proposed approach. While work like [4] explore the energy-delay trade-off using monolithic task scheduling, it lacks dependency awareness. Similarly, [15] proposes an energy-delay balanced placement strategy for multiple services without considering dependencies. More recently, [5] presents a task offloading scheme for dependent tasks, jointly optimising latency and energy. However, their focus is on local execution vs. edge offloading, while ours leverages both Multi-access Edge Computing (MEC) and cloud resources. Similarly, regarding the energy consumption optimisation in dependency-aware placement strategies, [13,19] primarily focus on minimising energy consumption at the user device level.

The conference-version of this work [27] further distinguishes itself by tackling a more realistic and complex Cloud-MEC scenario. To sum up, unlike approaches tied to specific architectures that can limit applicability, our methodology is designed for broader use. Furthermore, while many studies overlook inter-service dependencies or address them by focusing on single objectives or requiring techniques like service replication, our work directly incorporates these dependencies. We introduce two heuristic-based algorithms tailored for rapid execution, which simultaneously optimise both user-perceived latency (QoS) and total energy consumption (sustainability). Critically, whereas prior studies with similar objectives often focused on simpler local-vs-edge offloading decisions, our algorithms address the more granular and practical problem of detailed service placement within the MEC infrastructure alongside cloud offloading. This comprehensive approach, balancing multiple objectives while managing dependencies in a flexible architecture, highlights the novelty and practical significance of our contribution for system-wide optimisation in heterogeneous Cloud-MEC environments.

Integrating the energy harvesting technique into the MEC system is becoming a popular trend in the academia to pursue a sustainable operation of the micro datacenters. Work from [20] equipped EH devices at the base station to capture renewable energy as a complement for the grid hybrid electricity, and aim at minimising the electricity cost of base stations and the cloud offloading cost while maintaining the stability of the task queue. Work from [26] designed a partial task offloading strategy in energy-harvesting-enabled MEC, optimising both the task execution time and the grid energy cost. When the operation of the whole system solely rely on the harvested energy, the amount of available energy is limited, and the energy consumption of each device involved need to be properly monitored and managed to stay within the budget. In studies from [21,23], MEC servers and IoT devices are purely powered by the EH devices. [21] partitions individual tasks and select whether to process them partially on the IoT devices or offload them to the MEC servers, while no cloud computing resources are introduced. To process as many tasks at the edge as possible in a limited time period, and to minimise the fee charged by the cloud operator, [23] developed a meta-heuristics to assign the tasks to homogeneous servers and scales down the server operating frequencies to reduce energy consumption. [24] work on the dynamic task offloading for multiple terminal devices and edge servers in IIoT using Lyapunov optimisation, assuming that terminal devices harvest energy and store locally.

Our work addresses a more challenging and realistic problem formulation than typically found in the literature surveyed above, focusing on task graphs with general dependencies deployed across heterogeneous infrastructure. The complexity inherent in managing precedence constraints means prior methods for latency and deadline control are often no longer applicable. Effectively handling such dependencies is vital for accurately modelling and optimising many real-world workflows (e.g., complex data analytics pipelines, microservice chains). To meet this challenge, building upon [27], we introduce a novel online strategy that is capable of co-optimising efficient (renewable) energy usage and SLA compliance under these demanding conditions. The novelty is embodied in the integration of three: decomposing end-to-end deadlines into manageable sub-deadlines across dependency paths, employing dynamic frequency scaling at the edge, and offloading tasks to cloud adaptively. This approach provides a necessary advancement for scheduling complex, dependency-laden applications effectively in practical, heterogeneous Cloud-MEC systems.

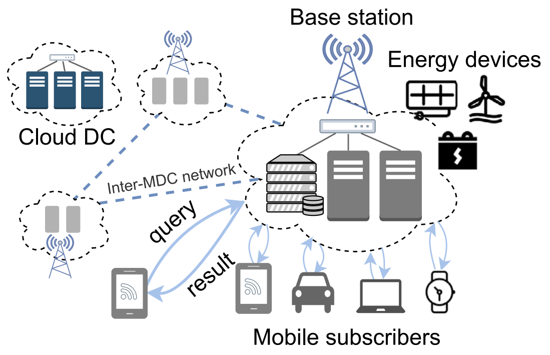


Fig. 1. An example Cloud-MEC network with energy-harvesting enabled.

3. System model and problem formulation

In this section, we formally describe the service placement and resource scheduling problem in the MEC system. The system offers the potential for low-latency processing close to users; however, with the integration of energy harvesting techniques, its operation may depend on variable energy sources (e.g., solar power) and finite battery storage at the edge. This introduces a fundamental challenge: the need to reduce service latency and meet strict deadline constraints (when they exist) often conflicts with the conservation of limited energy. Executing tasks faster or more frequently consumes more energy, potentially depleting reserves, whereas aggressive energy conservation can result in deadline violations. This creates a trade-off that strongly influences task placement decisions. Executing a task on edge nodes can offer low latency but is only feasible when sufficient harvested or stored energy is available; otherwise, scaling down server processing speeds or offloading tasks to the remote cloud becomes necessary, each impacting latency and energy consumption differently. Thus, effectively managing the interplay between deadlines, dynamic energy availability across heterogeneous nodes, and resource allocation strategies is essential to ensure both Quality of Service (QoS) and the sustainability of the EH-MEC system. The following models formalise the system variables, constraints, objectives, and decision variables. Table 2 presents the list of key symbols used in the formulation.

3.1. MEC network model

Our work considers a multi-tier Cloud-MEC network infrastructure where micro datacenters (MDCs) act as resource pools at the edge, residing at interconnected mobile stations. Within each MDC, two key entities exist: **computing servers** available for service deployment and execution, and **data sources** that provide data flow from sources like databases, sensors, and IoT devices without performing computations themselves. User equipment (UE) may connect to the local network of MDCs through radio access network (RAN) and submit service requests. The multi-access edge network is connected to a resource-rich cloud datacenter (Cloud DC). Fig. 1 shows an example of the Cloud-MEC system we consider in this work.

Each MDC maintains its energy harvesting (EH) devices that capture the renewable energy sources from the nature and supply the computing servers, as illustrated in the figure. The harvested energy will be stored in the local battery of the EH device, and the servers are able to consume the energy from the battery. Besides, if the harvested energy cannot satisfy the energy demand of MDC, extra energy can be purchased from the power grid. The cloud DC, on the contrary, is fully powered by the power grid.

We use $G = (M, C, L)$ to denote the Cloud-MEC network. The set of MDCs is expressed as $M = \{m_1, m_2, \ldots, m_n\}$, while C represents the Cloud DC. $L = \{l_1, l_2, \ldots, l_p\}$ is set of network connections between the MDCs. For each link in L, the transmission bandwidth is $bw(l)$, and the propagation delay is set to be a constant value $prop(l)$.

For every MDC m_i, the set of computing servers maintained by it is referred to as $S_i = \{s_1, s_2, \ldots, s_m\}$, and the data sources connected to the local network of m_i is expressed as $D_i = \{d_1, d_2, \ldots, d_k\}$. For each server $s_i \in S$, we symbolise the maximum processing frequency of its CPU as f_{s_i}. The Cloud DC charges based on the total amount of workload, i.e. number of operations, with a unit rate of cp. Inside the MEC network, we denoted the routing path between any pair of servers (s_i, s_j) as $path(s_i, s_j) = \{l_1, l_2 \ldots l_n\}$.

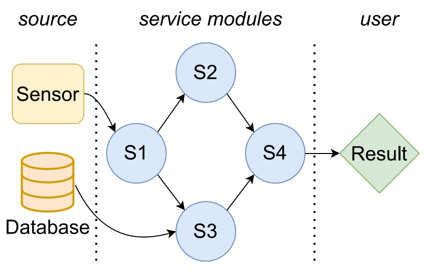


Fig. 2. An example application graph.

3.2. Application model

Our processing model adapts the directed acyclic graph (DAG) model. Let $A = (V, E)$ represent the application graph, where the set of application modules is denoted by $V = \{D, O, U\}$, and the set of directed edges representing dataflow dependencies is $E = \{e_1, e_2, \ldots, e_o\}$. $D = \{d_1, d_2, \ldots, d_m\}$ represents the data source modules, while $O = \{o_1, o_2, \ldots, o_n\}$ represents the service modules. The end receiver, referred to as U, is the user equipment. An example DAG application graph containing two source modules, four service modules and one end user can be seen in Fig. 2.

For any service module o, the amount of workload it requires to execute each task is denoted by $wl(o)$. We use $pred(o)$ and $succ(o)$ to denote the predecessor and successor modules of o.

Execution results will be generated and sent away by service modules through dependency edges. The application A will be triggered to process tasks continuously. Here we use $rate$ to define the task arrival rate of A. Besides, a deadline $deadline(A)$ may be set as the maximum acceptable task completion time of the application.

An application needs to be deployed on computing servers to run. Here we use $P(o)$ to represent the deployment plan of any service module o.

Our application model adheres to the following assumptions: (i) a service is triggered only when it has received inputs from all its predecessors, (ii) each task represents the minimal unit of work and is indivisible, and (iii) after finishing the task processing, a service node sends the corresponding results to all its successors simultaneously.

3.3. Problem formulation

3.3.1. Communication and computation model

The transmission delay T^{tran} for a data packet pkt through network link l can be calculated by:

$$T^{tran} = size(pkt)/bw(l) \tag{1}$$

The communication delay during this process is composed of the transmission delay and propagation delay:

$$T^{comm} = size(pkt)/bw(l) + prop(l) \tag{2}$$

For a directed edge e, when the start module e^{src} is offloaded to server s_i and the destination module e^{dst} is deployed on s_j, the delay of this edge is composed of the communication delay of each network hop it takes to forward a data packet pkt to the receiver:

$$T^{comm}(e) = \sum_{l \in path(s_i, s_j)} \frac{size(pkt)}{bw(l)} + prop(l) \tag{3}$$

Additionally, between two services deployed in the same MDC, we assume a constant network delay.

When computing server s_i runs at frequency f_{s_i} to process tasks, given the workload required by a service module o: $wl(o)$, the execution time can be calculated by:

$$T^{exec}(o, s_i) = wl(o)/f_{s_i} \tag{4}$$

We also define the overhead coefficient for the multi-tenancy scenario of n service modules deployed on one server: $K(n)$. When n service modules including o are co-located by one server, the task processing time of o becomes:

$$T^{exec}(o|n) = T^{exec}(o) * K(n) \tag{5}$$

For service modules with multiple predecessor nodes, the service module will not start the task execution until all its predecessors have finished their task processing and delivered the results to it. Therefore, for a service module o, the earliest time it starts task processing is decided by its most time-consuming predecessor. For o with its predecessors $pred(o)$, we obtain its *Earliest Start Time (EST)* by:

$$EST(o) = \max_{o_i \in pred(o)} \left\{ EST(o_i) + T^{exec}(o_i) + T^{comm}(o_i, o) \right\} \tag{6}$$

Similarly, the Earliest Finish Time (EFT), a value denoting the earliest time a service module o finish its task processing, can be obtained by:

$$EFT(o) = EST(o) + T^{exec}(o) \tag{7}$$

And therefore the end-to-end latency experienced by user U can be expressed as $EST(U)$.

When deadline constraints of application A exist, each service request sent to the system is supposed to be finished in a time no longer than $deadline(A)$. In such situation, we define the term Latest Start Time (LST): the latest point in time a service module can start without delaying the overall application or violating the deadline, and similarly the Latest Finish Time (LFT): the time point that a service can complete its task processing such that the overall latency in not lagged. Therefore for any module o, we can obtain recursively:

$$LFT(o) = \begin{cases} deadline(A) & \text{if o=U} \\ \min_{o_i \in succ(o)} \left\{ LST(o_i) - T^{comm}(o, o_i) \right\} & \text{otherwise} \end{cases} \tag{8}$$

and

$$LST(o) = LFT(o) - T^{exec}(o) \tag{9}$$

3.3.2. Energy consumption model

The main energy consumption of each edge computing server is expressed as the sum of static energy consumption and dynamic energy consumption. The dynamic power consumption of the CMOS circuits is more dominant and significant comparing to the static power consumption, and is our main focus. Given the supply voltage of the CMOS circuits v and the CPU operating frequency f, the dynamic power consumption can be obtained by: $P = kv^2 f$, where k represents a device-related coefficient. Here we assume that v and f are linearly dependent, therefore: $P = \beta f^3$, where β represents a device related factor.

We drive the dynamic energy consumption of each server s by calculating its dynamic power consumption over time, expressed by:

$$EC(s) = \int P_s \, dt = \int \beta_s {f_s}^3 \, dt \tag{10}$$

During the processing of a type o task, if the CPU frequency remains unchanged, combining with (4), the total dynamic energy consumption can be approximated by:

$$EC(s, o) = \beta_s {f_s}^2 wl(o) \tag{11}$$

Below we discuss the energy models surrounding the energy-harvesting devices. For an energy harvesting device EH_{m_i} power MDC m_i, we use $P_{harvest}$ to denote its charging power. The total instant power consumption of MDC m_i can be expressed as:

$$P_{m_i} = \sum_{s \in S_i} \beta_s {f_s}^3 \tag{12}$$

Similarly, the total energy consumption of MDC m_i is:

$$EC(m_i) = \sum_{s \in S_i} \int \beta_s {f_s}^3 \, dt \tag{13}$$

3.4. Battery model

The energy harvesting device powering a MDC m is denoted by EH_m, and the energy storage capacity of EH_m's battery is expressed as $Capa(EH_m)$. We partition the time into a continuous series of time slices $1, 2, 3, \ldots, n$, with equal lengths Δt. A new task may be generated in each time slice. We use $B_m(k)$ to represent the remaining power in the battery of the energy harvesting device EH_m at the beginning of the kth time slice.

During the kth time period, assuming that the operating frequency of each active server remain constant, we can drive that:

$$B_m(k+1) = B_m(k) + E_m^{harvested}(k) - E_m^{consumed}(k) \tag{14}$$

Also, the remaining power in the battery must satisfy the following at any time:

$$0 \le B_m(k) \le Capa(EH_m) \tag{15}$$

Use B_m^{ini} to denote the initial remaining power of battery B_m. Since MDCs purely rely on the harvested energy, combining with Eq. (14), the energy stored in B_m at the beginning of time slot k is:

$$B_m(k) = B_m^{ini} + \sum_{n=1}^{k-1} (E_m^{harvested}(n) - E_m^{consumed}(n)) \tag{16}$$

Since the energy consumed by all MDC servers during the time slice cannot exceed the energy stored in the battery, we also have: $E_m^{consumed}(k) \le B_m(k)$.

In scenarios where the harvested energy is insufficient to support the task processing during one time slice, the cloud offloading will be triggered.

3.5. Cloud offloading and cost models

If tasks to be processed by service o are offloaded to the cloud, with knowledge of the workload demands per task and the unit pricing of the cloud service, the computational cost to process one task can be expressed as:

$$Cost_o^{cloud} = wl(o) * cp \tag{17}$$

During the kth time slice, let $tackle_j^k$ denote the total number of subtasks processed by service module o_j. Cloud offloading for service module o_j may be triggered at certain time points, incurring cloud execution costs. We use the binary variable x_{ij}^k to indicate whether the ith subtask (where $i = 1, \ldots, tackle_j^k$) is executed on the cloud. If the subtask id executed on the cloud, $x_{ij}^k = 1$; otherwise, $x_{ij}^k = 0$. By summing up the costs corresponding to the computation offloaded to the cloud, the total cloud fee for o_j during the kth time slice can be expressed as:

$$Cost^{cloud}(k) = \sum_{j=1}^{tackle_j^k} \sum_{i=1}^{n} x_{ij}^k * wl(o_i) * cp \tag{18}$$

3.6. Placement problem formulation

The objective of this problem is to meet the service latency deadline while maximising renewable energy utilisation and minimising total cost. The solution should be an appropriate initial mapping from the service nodes to the servers, followed by server frequency scaling or module offloading decisions that will be made dynamically.

We call a placement plan valid only if all the service modules are assigned to servers with sufficient resources and all the constraints are met. In light of this, we formulate the conditions that make a placement plan valid and our optimisation goals:

(i) Given the set of service modules O to place and the set of available servers in the Cloud-MEC environment G, each service in O should be mapped to a server in G. We formulate such a progress as a mapping function $Placement$, let S denote the set of all servers in the network, then $Placement : O \rightarrow S$ indicates a complete deployment of all services.

(ii) As explained in [27], provided that (i) is satisfied and the summation of resource occupancy for all assigned modules does not violate the capacity of the server, the optimisation objective is to minimise the user-experienced latency and the overall dynamic energy consumption jointly, formulated as:

$$P_1 : min\ EST(U), min \sum_{s \in S} EC_s \tag{19}$$

(iii) If the deadline of an application is pre-defined in the service-level agreements, it must not be violated, therefore the tackle time for every task arrived at the kth time slice should always be within the deadline: $T_{delay} \leq deadline(A)$.

(iv) In the energy-harvesting scenario, provided that (i) and (iii) are satisfied, the optimisation goal to minimise the cloud fee can be expressed as:

$$\begin{aligned} P_2 :& min \sum_{j=1}^{tackle_j^k} \sum_{i=1}^{n} x_{ij}^k * wl(o_i) * cp, \\ s.t.&(i),(ii) \\ & 0 \leq B_m(k) \leq Capa(EH_m), m \in M \\ & 0 \leq f_s(k) \leq f_s, s \in S \\ & x_o(k) \in \{0,1\}, o \in O \\ & E_m^{consumed}(k) \leq B_m(k), m \in M \end{aligned} \tag{20}$$

4. Energy-and-latency-aware placement algorithms in MEC

4.1. Energy-aware delay-experienced minimisation algorithm

The proposed service placement algorithm, energy-aware delay-experienced minimisation (EDEM), adopts a two-stage approach to achieve a balance between latency minimisation and energy consumption reduction. The flow of the EDEM algorithm is illustrated in Figs. 3, 4, 5. The coarse-grain stage (Figs. 3 and 4) runs first, followed by the fine-grain stage (Fig. 5). In the coarse-grained stage, EDEM determines a service-to-MDC deployment plan that prioritises reducing end-to-end latency. Subsequently, the fine-grain stage refines the server deployment plan within each MDC, focusing on optimising energy consumption without affecting the overall latency.

Coarse-grain scheduler: Base on the critical path method (CPM), the coarse-grain scheduler seeks a balanced configuration with minimal transmission latency and maximised processing efficiency, leading to the lowest approximated end-to-end latency. The pseudo-code of the coarse-grain scheduler is presented in Algorithm 1.

We use $S(A)$ to denote the state space of services in application A, consisting of all service-to-MDC placement options. The scheduler explores $S(A)$ by post-order traversing all placement options of the service modules in A. According to Eq. (6), the calculation of EST for any service v rely on the EST value of all its predecessor modules $v.predecessors$. Following such rule, the scheduler can estimate the EST value for each service, paving the way for critical-path selection.

The critical path selection works in a greedy and optimistic fashion. As shown in Fig. 3, the bottom-up latency approximation identifies a critical path, highlighted with a red stroke, from the original application graph. Starting with the bottom-most modules that directly interface with the data sources, for each visited service v, given the set of available MDCs M, the value of $EST(v)$ will be approximated assuming v resides at the least-loaded server $m.leastloaded$ in each MDC

Algorithm 1: EDEM

```
Data: App. $A = (V, E)$, MEC $G = (M, L)$
Result: Server placement map $P : O \rightarrow S$
1. Coarse-grain stage:
Initiate $S(A)$ $\forall v \in V, \forall m \in M$; $CP = \emptyset$;
Explore $S(A)$ using post-order traversal:
for $v.pred \in v.predecessors$ do
    for $m \in M$ do
        Compute $EST(v.pred,m)$;
    end
    Select $(v.pred,m)$ with $min\ EST(v.pred,m)$;
end
CP $\leftarrow$ $(v.pred,m)$ with $max(EST(v.pred,m))$;
Compute $Criticality(v)$;
2. Fine-grain stage:
for $(v, m) \in critical\_path$ do
    Initiate $S(V)$ $\forall s \in m.servers$;
    for $s \in m.servers$ do
        if $EST(v, s) \leq Criticality(v)$ then
            Compute $EC(v,s)$;
    end
    Assign $v$ to $s$ with $min\ EC(v,s)$;
    Update load status of $s$;
end
for $(v, m) \notin critical\_path$ do
    Initiate $S(V)$ $\forall s \in m.servers$;
    for $s \in m.servers$ do
        if $EST(v, s) + T_{comm}(v, v.succ) \leq Criticality(v.succ)$ then
            Compute $EC(v,s)$;
    end
    Assign $v$ to $s$ with min EC(v,s);
    Update load status of $s$;
end
```

Table 2
Table of key notations

Indices	Description
$G = (M, C, L)$	Cloud-MEC network
$M/C/L$	Set of MDCs /Cloud DC/network connections
$bw(l), prop(l)$	Bandwidth and propagation delay of l
S_i	Computing servers in MDC m_i
f_{s_i}	Maximum CPU frequency of server s_i
cp	Unit price of cloud DC
EH_m	EH device of MDC m
$Capa(EH_m)$	Battery capacity of EH_m
$EC(s)$	Energy consumption of server s
$path(s_i, s_j)$	Routing path between two servers
B_m	Battery level of EH device of MDC m
A	Application graph
$S(A)$	State space of services in A
D/O	Set of data sources/service modules in A
$wl(o)$	Workload of single task processed by module o
E	Set of dependency edges in A
$pred(o), succ(o)$	Set of predecessor and successor modules of o
$deadline(A)$	Maximum task completion time of A
$sd(o)$	Sub-deadline of module o
$EST(o)$	Earliest processing start time of module o
$LST/LFT(o)$	Latest processing start/finish time of module o
$AST/AFT(o)$	Actual processing start/finish time of module o

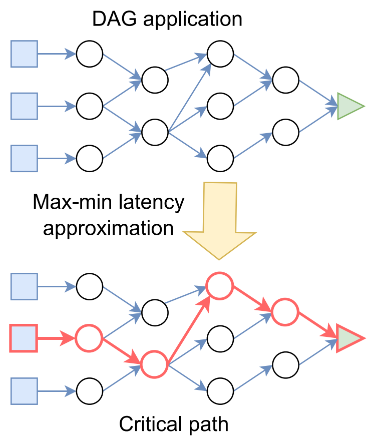


Fig. 3. The coarse-grain stage of EDEM: critical path selection.

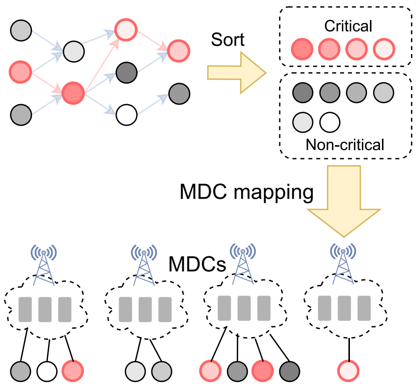


Fig. 4. The coarse-grain stage of EDEM: service-to-MDC mapping.

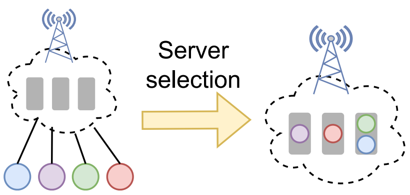


Fig. 5. The fine-grain stage of EDEM: service-to-server mapping.

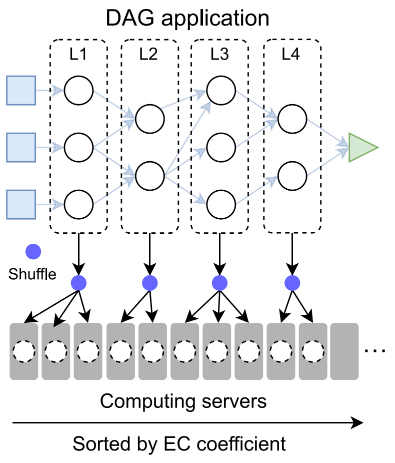


Fig. 6. The phase one of DEM: greedy initialisation.

$m \in M$. For each predecessor $v.pred$ of v, knowing its estimated EST values $[EST(v.pred, m_1), EST(v.pred, m_2), \ldots]$, the predecessor-MDC pair $(v.pred, m)$ yielding the minimum estimated EST is identified. Among all the selected predecessor-MDC pairs, the pair with the maximum EST is designated as the critical node. Subsequently, the criticality of non-critical modules within the same hierarchical level is determined by calculating the EST difference between the module and the identified critical node. This process is iteratively repeated until the final exit node of the application is reached.

The coarse-grain stage of the algorithm goes on as illustrated in Fig. 4. The previous phase establishes the critical path, aligning with the MDC allocation plan for the critical modules. Non-critical modules remain without assigned MDCs at this point. Subsequently, the load status of the MDCs participating in the critical path allocation plan is updated, as is the least-loaded server $m.leastloaded$ within each MDC $m \in M$. The max–min procedures then resume, focusing on allocating non-critical modules until their MDC assignments are finalised. With the service modules mapped to different MDCs, as indicated by black links in the figure, the algorithm proceeds to the next step of module-to-server allocation.

Fine-grain scheduler: Following the determination of the service-to-MDC placement plan during the coarse-grained stage, for each MDC involved, a fine-grain scheduler runs to drive the module-to-server placement solution. As can be seen in Fig. 5, in this stage, the server selection for service modules is performed independently within each MDC. The fine-grained scheduler prioritises placing modules on less power-consuming servers while ensuring that the end-to-end latency of critical nodes remains unaffected. Critical modules will take priority in placement, followed by non-critical modules. The pseudo-code of the fine-grain scheduler is presented as Step 2 in Algorithm 1. Similar to the design of the coarse-grain scheduler, the fine-grained scheduler thoroughly evaluates all service-to-server placement options. For each module and every available server within the local MDC, the scheduler estimates the earliest start time. Subject to the constraint that the EST value of modules on non-critical branches must not exceed the EST of critical modules within the same hierarchical level, we evaluate the EST value $EST(v, s)$ for each potential placement option (v, s). Placement options that violate this constraint are discarded, while the remaining options are deemed *valid*. From these valid options, the placement with the lowest energy consumption $EC(v, s)$, as approximated by Section 3.3.2, is selected and incorporated into the placement plan. Upon finalising a service placement decision, the load status of the designated server is updated, and the estimated EST values of all affected placement options are recalculated. This iterative process continues until all service modules are assigned to specific servers.

4.2. Delay-Aware Energy Minimisation algorithm (DEM)

In contrast to EDEM, which prioritises latency minimisation, the proposed delay-aware energy minimisation (DEM) algorithm prioritises energy efficiency. DEM initially seeks an energy-saving server placement plan. Subsequently, a refinement stage fine-tunes this plan to optimise latency while strictly adhering to established energy constraints. DEM can be broken down into the 3 steps, in order as shown in Figs. 6, 7, 8:

Step 1: Greedy initialisation. All available servers within the MEC network are sorted by their device-related energy consumption coefficients, in ascending order, as illustrated in Fig. 6. Then a level-order traversal of the application graph starts from the entry modules. At each layer of the traversal, as split with dashed box in the figure, modules are randomly assigned resources from the pre-ranked server list. Following the assignment, DEM estimates the EST value for each module, using Eq. (6).

Step 2: Service-to-MDC placement alteration. While ***Step 1*** prioritises energy-efficient servers, it might not necessarily achieve the

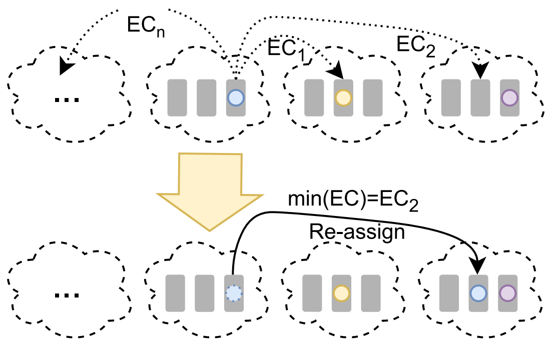


Fig. 7. The phase two of DEM: Adjust the service-to-MDC mapping.

absolute minimum total energy consumption. This is because geographically distant placements of interdependent modules can result in prolonged data transmission times and increased idle energy consumption at servers awaiting packets. To address this, a refinement stage iteratively explores alternative placements for each service module, as presented in Fig. 7. For each module, all unexplored MDCs (excluding its current location) are considered. Within each unexplored MDC, the least power-consuming server is evaluated as a potential reassignment target. As indicated by the dotted arrows, the algorithm estimates the total energy consumption EC after each potential reassignment, and updates the placement plan if a more energy-efficient configuration is identified. Finally, the refined placement's overall latency $EST(U)$ and total energy consumption $\sum_{s\in S} EC(s)$ are calculated and stored for the next stage.

Algorithm 2: DEM

```
Data: Application $A = (V, E)$, MEC network $G = (M, L)$
Result: Server placement map $P : O \to S$
1. Server sorting and initial service assignment:
Sort all servers $s \in G$ by $coeff(s)$;
$curServerIdx \leftarrow 0$;
Group service from level $n$ to 0 and shuffle each set $S_n, ...S_0$
for service $v \in S_n, ..., S_0$ do
    Assign $v$ to server with index $curServerIdx$;
    $curServerIdx{+}{+}$;
end
for $v \in V$ do
    Compute $EST(v,P(v))$ using Eqs;
end
2. Energy-aware reassignment:
for $v \in V$ do
    for $m \in M$ do
        if $P(v) \notin m$ then
            $s = \arg\min_{s' \in m.\mathrm{servers}} \mathrm{coeff}(s')$;
            if $EST(v,s)<EST(v,P(v))$ then
                Assign server $s$ to $v$;
        end
end
Compute $EST(U)$ using Eqs;
Sum up $Estimated EC(v, P(v))\ \forall\ v \in V$;
3. Latency-aware reassignment:
Identify $critical\_path$ of A under placement $P$;
for $(v, P(v)) \in critical\_path$ do
    for $s \in P(v).mdc.servers$ do
        if $EST(U) > EST(U|P(v) = s)$ then
            Assign server $s$ to $v$;
            Re-identify $critical\_path$ of $A$;
    end
end
```

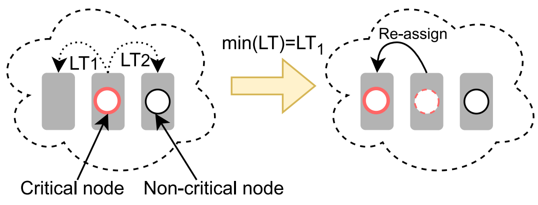


Fig. 8. The phase three of DEM: Fine-tune the service-to-server mapping.

Step 3: Service-to-server placement fine-tuning. Building upon the refined resource allocation from (***Step 2***), DEM also focuses on improving user-experienced latency. Similar to EDEM's fine-tuning approach, it identifies critical modules within the application's critical path. Fig. 8 illustrates this phase of DEM. For each critical module, potential reassignments to alternative servers are explored with the aim of reducing critical path latency. For each reassignment option, indicated by a dotted arrow, the end-to-end latency is re-approximated and compared against that of the current assignment plan. After each reassignment, the critical path is recalculated to assess potential latency improvements. This iterative process continues until no further latency reductions are achievable. The resulting placement plan, effectively balancing energy efficiency and latency, is then utilised to generate the final module-to-server mapping. The pseudo-code of DEM is presented in Algorithm 2.

5. Dynamic resource scheduling and service offloading (DSO) algorithm in EH-MEC

After integrating the energy harvesting technique with the MEC system, deadline constraints and energy budgets must be considered throughout the application execution.

Therefore, with the two offline placement algorithms in hand, we decided to design an online scheduling strategy that monitors the status of the MEC-EH system and continuously makes decisions for server operating frequency scaling and cloud offloading. Our online scheduling algorithm can be divided into three phases:

Phase 1. Initial energy-latency-aware service placement. At the initialisation stage, we pass the edge network information and the application to the EDEM or DEM algorithm to generate a service-to-server assignment plan. Service modules will be deployed on MEC edge servers, after which application execution will begin.

Phase 2. Deadline constraints breakdown. Given the deadline constraint of the application, and with knowledge of the app graph structure as well as the task size and workload of every service, we calculate a sub-deadline for each module. Inspired by the deadline decomposition method from [28], we traverse all modules in a top-down order and determine the value of $LST(o)$ and $LFT(o)$ of each module $o \in O$ according to Eqs. (8) and (9). Starting from the overall deadline and working backward, we use the upward rank $rank(o)$ to quantify the impact of the module to the entire schedule: the longer it takes to process sub-tasks and transmit data from the module to the end user, the higher the $rank(o)$ value becomes. Finally, The sub-deadlines of every module o, denoted as $sd(o)$, are calculated as proportions of $deadline(A)$.

The relations between the rank of a module o and its successor modules can be expressed as:

$$rank(o) = \max_{o_i \in succ(o)} \left\{ rank(o_i) + T^{comm}(o, o_i) \right\} + T^{exec}(o) \tag{21}$$

Therefore the highest rank belongs to the entry source modules, calculated as:

$$rank_D = \max_{d_i \in D} \left\{ rank(d_i) \right\} \tag{22}$$

and the sub-deadline of module o can be calculated as follows:

$$sd(o) = \frac{rank_D - rank(o) + \min_{s_i \in S}\left\{T^{exec}(o, s_i)\right\}}{rank_D} * deadline(A) \tag{23}$$

These sub-deadlines indicate the maximum flexibility each service has before impacting the overall schedule, providing room for further resource optimisation and scheduling.

Algorithm 3: DSO

```
Data: Application $A = (V, E)$, $deadline(A)$, EH-MEC network
      $G = (M, L)$
Result: Server placement map $P : O \to S$, scaling and offloading
        decisions
1. Initial service assignment using our algorithms:
$Algorithm(A, G) = P : O \to S$
for $o \in O$ do
    Deploy $o$ to server $P(o)$;
end
2. Deadline constraints breakdown:
Explore $A$ using level-order traversal:
for $v \in A.current\_level$ do
    Compute $LST(v)$ and $LFT(v)$;
end
for $v \in V$ do
    Compute $rank(v)$ and $sd(v)$;
end
3. Server scaling and service offloading:
while $A.active$ do
    for $m \in M$ do
        if $B(m) == 0$ then
            for $s \in m.servers$ do
                $O_{offload} \leftarrow \{o | P(o) = s\}$;
            end
    end
    for $o \in \{O - O_{offload}\}$ do
        if $AFT(o) - AFT(o.pred) > sd(o)$ then
            $O_{offload} \leftarrow \{O_{offload} + o\}$;
        else
            Compute $ratio(P(o))$;
            $P(o).scale\_down()$
        end
    end
    for $o \in O_{offload}$ do
        Undeploy $o$ from server $P(o)$;
        Redeploy $o$ to cloud $C$;
    end
end
```

Phase 3. Scaling and offloading decision making. The third phase is the online resource scheduling process, which runs continuously as the application actively receives new data. The scheduler monitors the system's real-time status, including the battery levels of energy-harvesting devices and the latency between service modules. We use $AFT(o)$ to denote the actual finish time of service module o, and use $\mathbb{E}(FT(o))$ to represent the expected finish time of o. According to the deadline constraints breakdown in Phase 2, we have:

$$0 \le \mathbb{E}(FT(o)) \le sd(o) \tag{24}$$

When the actual finish time of a service o is earlier than its sub-deadline, slack time is available. This creates an opportunity to scale down the operating frequency of the edge server, reducing energy consumption within an acceptable range and avoid draining the battery. The service time needed for o to process a task can be calculated by the difference between its actual start time and actual finish time, expressed as:

$$T^{exec}(o, P(o)) = AFT(o) - AST(o) \tag{25}$$

We use $ratio(s_i)$ to denote the scaling ratio of the CPU operating frequency of server s_i, and combining Eq. (24) we have:

$$AFT(o) < \mathbb{E}(FT(o)) \le sd(o) \tag{26}$$

Assume the finish time of service o equals the sub-deadline, regarding the theoretical minimum allowed operating frequency we have:

$$wl(o)/f^{min} = sd(o) - AST(o) \tag{27}$$

and so the theoretical minimum allowed value of the scaling ratio equals:

$$ratio(s_i)^{min} = \frac{f^{min}}{f(s_i)} = \frac{AFT(o) - AST(o)}{sd(o) - AST(o)}, P(o) = s_i \tag{28}$$

Taking the midpoint between the minimum and maximum scaling ratio we get:

$$ratio(s_i) = \frac{sd(o) + AFT(o) - 2AST(o)}{2 * (sd(o) - AST(o))}, P(o) = s_i \tag{29}$$

and the operating frequency of server s_i will be scaled down by $ratio(s_i)$.

Conversely, when the actual finish time of a service o exceeds its sub-deadline, or the battery of the local EH device is depleted, cloud offloading for the service module is triggered. The module will be undeployed from the MDC and redeployed to the cloud, at which point cloud billing will commence.

The pseudo-code of the strategy is presented in Algorithm 3.

6. Experimental evaluation

6.1. Performance indicators & setup

We employed simulation to evaluate the performance of the proposed placement algorithms (DEM and EDEM) and the dynamic resource scheduling algorithm (DSO). The YAFS fog simulator [6] was used and extended to support sequential processing of dependent tasks. The edge network topologies were created using the *NetworkX* library[2] with different random graph generation models: Barabasi–Albert (B-A) [29], Watts–Strogatz (W-S) [30], and ring topology.

To obtain more generalisable results, real-world workloads were employed from the Alibaba cluster trace dataset.[3] This dataset provides DAG information of production batch workloads from a large-scale cluster. To model the user-submitted jobs consisting of services such as video stream processing, image recognition, machine learning inference, real-time analytics, and large-scale data preprocessing, we randomly selected 10 applications for evaluation from those exceeding 10 modules, the characteristics of which are listed in Table 4. Module resource requirements were configured based on the provided traces. Within the simulation's temporal settings, tasks arrive periodically, with a constant interval of 100 global timestamps, as new data continuously flows in from IoT sensors. For each experiment set, system events were simulated for 20,000 global timestamps and repeated 5 times with identical configurations to generate statistically significant averages. The simulations were conducted on a server with 4x Intel Xeon Gold 6230N CPU, 256 GB of RAM and Ubuntu 20.04 operating system.

6.1.1. Service placement experiment setup

To evaluate the proposed service placement algorithms (DEM and EDEM), we constructed heterogeneous MEC-Cloud topologies consisted of up to 20 micro edge datacenters and one cloud datacenter using the three different models mentioned above. Each MDC housed up to 5 computing servers. Details regarding the specific MEC network configurations are provided in Table 3. To model a heterogeneous computing environment, we configured distinct resource characteristics for cloud and edge servers.Cloud servers, representing powerful centralised resources, were assigned CPU frequencies of 3–5 GHz and RAM of 32–64 GB, reflecting common virtual machine instance types in public clouds (e.g. Amazon EC2, Azure and Google Cloud). In

[2] https://networkx.org/.
[3] https://github.com/alibaba/clusterdata.

Table 3
MEC-Cloud environment configurations.

	Cloud	MDC
CPU frequency (GHz)	[3,5]	[1,2]
RAM (GB)	[32,64]	[1,4]
Propagation delay (ms)	8	1
Bandwidth (Gbps)	10	2

Table 4
Application characteristics.

Job id	$\|V\|$	$\|E\|$	Max degree	Average transfer volume	Average workload
1	12	9	3	39.33	9.50
2	16	17	6	29.00	257.88
3	16	17	7	23.63	40.50
4	17	17	5	42.82	13.53
5	16	17	3	46.06	35.75
6	12	11	6	38.67	8.17
7	10	10	4	44.30	14.40
8	10	9	5	35.60	7.10
9	16	16	2	47.19	1.00
10	10	9	4	43.40	32.70

contrast, edge servers featured more constrained resources (1–3 GHz CPU, 1–4 GB RAM), typical of edge hardware deployed closer to users under power and cost limitations, including ARM-based platforms and compact x86 systems. Network connectivity also differed: the link to the cloud had high bandwidth (10 Gbps) but also higher latency (10 ms one-way delay), representing a WAN connection over larger geographical distances. The edge network link offered lower bandwidth but minimised latency, characteristic of local access networks like 5G or LANs. These parameters were chosen to create a realistic testbed contrasting centralised and edge capabilities.

The two algorithms were evaluated based on four key metrics: (a) Overall Energy Consumption (EC): This metric represents the total power consumed by all servers during the execution period, estimated from CPU usage data according to 3.3.2. (b) Average User-Experienced Latency (LT): This metric is the mean response time of user requests, calculated from raw timestamps recorded by the simulator during network transmissions. (c) Edge prioritisation (EP): This metric reflects the percentage of services deployed at the Edge. (d) Algorithm Execution Time (ET): This metric indicates the time required for the algorithm to generate a placement, i.e. the wall clock time for executing each algorithm. The performance of DEM and EDEM was compared against four state-of-the-art service placement algorithms: Response Time Aware (RTA) [16], Genetic Algorithm (GA) [31], Maximise Reliability Offloading (MROA) [19] and Energy-Makespan Multi-objective Optimisation (EM-MOO) [22]. We selected these algorithms for their varying levels of complexity, computational overhead, and optimisation goals. By comparing our proposed algorithms with these diverse alternatives, we gain a broader understanding of their strengths and weaknesses, enabling a more robust assessment of their overall performance and efficiency.

6.1.2. Dynamic scheduling experiment setup

To evaluate the performance of our online resource scheduling strategy (DSO), unlike the previous experimental settings, we constructed MEC-Cloud environments featuring energy-harvesting-enabled base stations. The network topology generation used the Barabasi–Albert (B-A) model, and each MDC is purely powered by an energy harvesting device with a battery storage. Detailed configurations of the MEC-EH environment are shown in Table 5, similar to Table 3, with specs commonly observed in cloud and edge computing scenarios. The algorithm receives the result from either the EDEM or DEM algorithm as the initial service placement plan and makes server frequency scaling or service offloading decisions dynamically.

We evaluated the algorithm based on five key metrics: (a) Number of Processed Requests (TP): the amount of use requests that have been successfully processed, recorded after the receiving of each result at the user side. (b) Average User-Experienced Latency (LT): the mean response time of user requests processed, extracted from the message timestamps stored in the simulation traces. (c) The Acceptance Rate of Processed Requests (AC): the proportion of successfully processed requests that meet the deadline constraint, recorded after the arrival of results. (d) Overall Energy Consumption (EC): the total power consumed by edge and cloud datacenters during the task processing, estimated from CPU usage history using 3.3.2. (e) Cloud Renting Cost (CRC): the total fee charged by the cloud datacenter for task execution, calculated based on billing time.

Table 5
EH-enabled MEC-Cloud environment configurations.

	Cloud	MDC-EH
CPU frequency (GHz)	4	[1,2.5]
Propagation delay (ms)	50	5
Bandwidth (Gbps)	10	1
Renting fee ($ per s)	0.01	N/A
Battery capacity (Wh)	N/A	10
EH device charging rate (W)	N/A	1
Device power factor	10^{-27}	10^{-27}

We compared the performance of DSO-DEM and DSO-EDEM against three different scheduling options: (a) $offloading$, a policy that dynamically offloads every service of the application to the Cloud, (b) $scaling$, a policy that makes server frequency scaling if there is enough slack time between service's finish time and sub-deadline, and (c) $none$, a static baseline policy that takes no subsequent action after the initial placement. For all scheduling variations, we assume that each datacenter is connected to a renewable energy source and is equipped with a battery storage. Green energy will be harvested to power the datacenter continuously. We aim to evaluate how different scheduling strategies impact user experience, energy usage and operational costs through our experiments.

6.2. Performance assessment

6.2.1. Service placement algorithm evaluation

To evaluate the performance of the EDEM and DEM algorithm, a total of 8100 experiments were conducted. For each of the 10 applications, 5 experiments were performed for each of the 6 algorithms. The number of MDCs (denoted by n) was varied across [5, 10, 20]. Similarly, the number of servers (denoted by m) within each MDC was varied across [2, 4, 8]. Furthermore, three different network topologies were utilised in the experiments. All figures presented in this section demonstrate the normalised performance of both EC and LT metrics. We set the source data emission interval to 100 ms and the simulated duration to 100 s and plot the performance of the proposed algorithms in terms of the aforementioned metrics. In Fig. 9, we show the normalised energy consumption results for all algorithms.

It can be seen that during the experiments across various applications and network topologies, the DEM algorithm achieves significantly lower energy consumption compared to other algorithms. In contrast, RTA and MROA exhibit the highest energy consumption. RTA, by primarily focusing on minimising end-to-end latency, often selects less energy-efficient servers, resulting in higher overall energy consumption. MROA, on the other hand, primarily focuses on reducing energy consumption at the user equipment, neglecting the energy consumption of the servers involved in task offloading, leading to high overall energy consumption for task processing. DEM, on the contrary, tends to allocate services to edge servers with higher energy efficiency. Following DEM, EDEM also achieves energy consumption levels comparable to GA and EM-MOO. Across diverse network topologies, DEM and EDEM exhibit stable performance in terms of normalised energy consumption, while EM-MOO falter under the Watts–Strogatz network model. This proves that DEM and EDEM are robust to network variations.

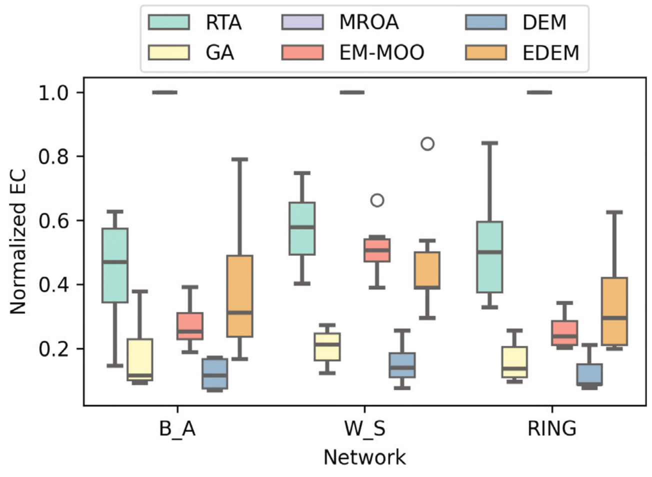


Fig. 9. Normalised Energy Consumption (EC), $n = 20$ $m = 4$.

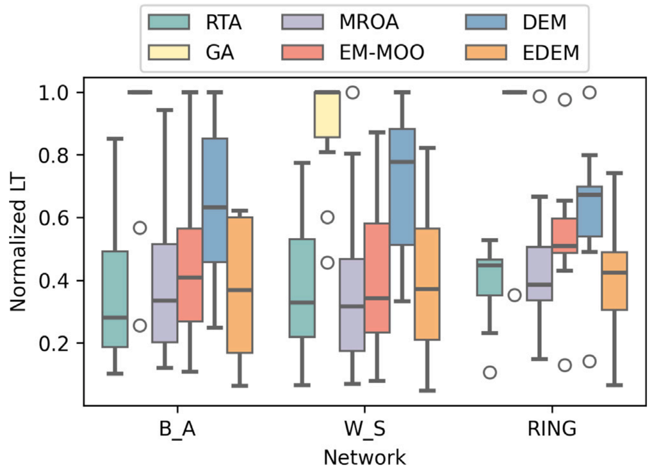


Fig. 10. Normalised user-experienced latency (LT), $n = 20$ $m = 4$.

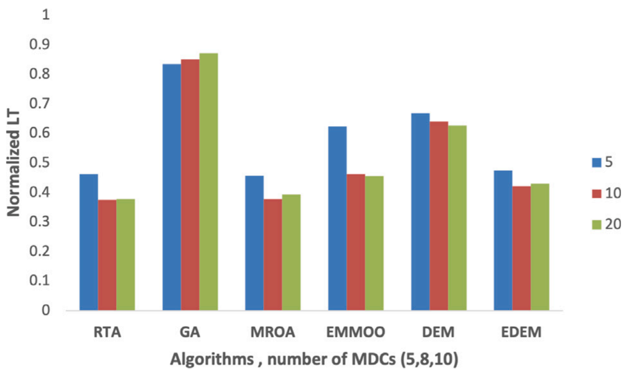


Fig. 11. Normalised user-experienced latency (LT), $n = [5,10,20]$.

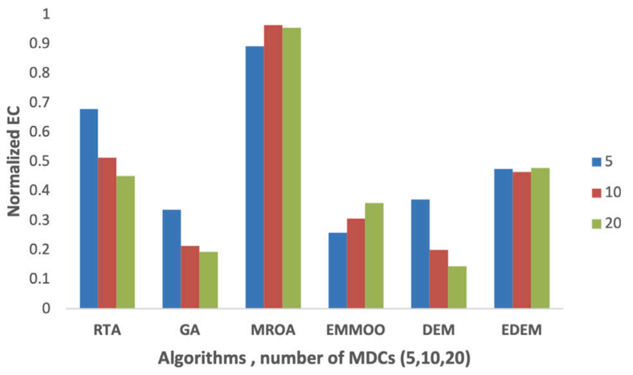


Fig. 12. Normalised Energy Consumption (EC), $n = [5,10,20]$.

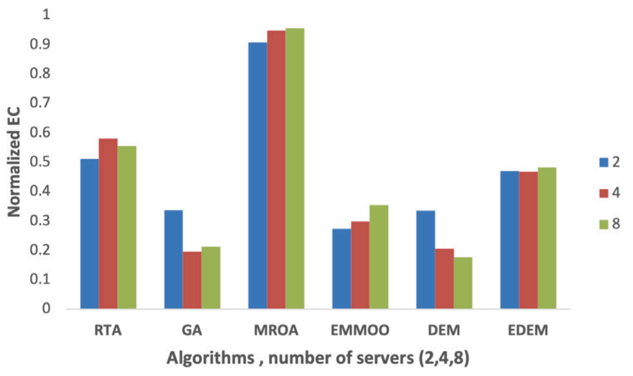


Fig. 13. Normalised Energy Consumption (EC), $m = [2,4,8]$.

Fig. 10 illustrates the normalised latency performance across all algorithms. RTA, EDEM, and MROA consistently achieve the lowest latency across all scenarios. However, it should be highlighted that RTA and MROA achieve these low latency values at the expense of significantly higher energy consumption, whereas EDEM maintains metrics. GA generally exhibits the poorest performance for all network topologies while EDEM excels in certain applications. Particularly, in the ring topology, where network latency can be a significant bottleneck, EDEM demonstrates superior latency performance. Combining the results from Figs. 9 and 10, it becomes evident DEM and EDEM exhibit distinct preferences in optimising for energy consumption and latency. EDEM effectively balances overall energy consumption with latency performance. On the other hand, DEM prioritises energy-saving placement plans,potentially resulting in some sacrifice of latency performance.

Figs. 12, 13 demonstrate that for the energy consumption metric, the DEM and GA algorithms show the most significant improvements and efficiency gains as the number of MDCs or servers increases, while MROA consistently underperforms. EMMOO shows a slight decrease in energy consumption as the number of resources grows. While not the most energy-efficient, EDEM demonstrates stable performance, making it a viable option. For the latency metric, RTA, EMMOO, and DEM perform better when increasing the number of MDCs, with EDEM following closely behind (Fig. 11). In contrast, GA and MROA struggle to effectively reduce latency as the system complexity increases.

In Fig. 14, we observe that increasing the number of servers generally leads to a degradation in latency performance for most algorithms. Although GA shows some improvement as the number of servers increases, it still fails to effectively optimise latency as the system scales up compared to other algorithms. In contrast, EDEM, mirroring its stable performance in terms of energy consumption, maintains a consistent level of performance. This makes EDEM the most suitable candidate when the number of servers per MDC exceeds four. These findings suggest that both EDEM and DEM exhibit good scalability and maintain stable performance as the system scales and the network topology expands.

Table 6 presents the percentage of services deployed at edge servers, providing insights into the service placement strategies of different algorithms. The results of MROA align with our analysis, demonstrating a preference for non-local, powerful machines. Consequently, around 40% of the services are offloaded to the cloud, leading to the highest observed energy consumption. RTA and EDEM have similar preferences in resource allocation, deploying 75%–85% of services at

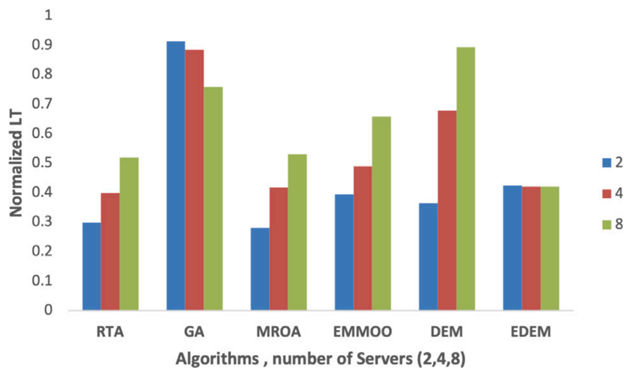


Fig. 14. Normalised user-experienced latency (LT), m = [2,4,8].

Table 6
Edge prioritisation, EP.

	RTA	GA	MROA	EMMOO	DEM	EDEM
B-A	85.86	97.75	58.52	98.80	98.24	85.70
RING	76.61	89.79	57.90	99.66	95.19	84.24
W-S	76.85	89.69	60.40	99.13	94.04	84.46

Table 7
Execution time (ms), ET.

	RTA	GA	MROA	EMMOO	DEM	EDEM
B-A	45.35	4110.22	3.13	1077.88	224.91	138.63
RING	42.79	4475.50	3.37	1066.64	315.66	173.59
W-S	45.96	4407.99	3.15	1064.44	290.66	161.51

edge facilities and the remaining tasks in the cloud. Such approach achieves a well-balanced outcome in terms of both latency and energy consumption by minimising transmission and energy costs while leveraging cloud resources for computationally intensive tasks. In contrast, DEM, EMMOO, and GA exhibit a strong preference for edge resource utilisation, deploying over 95% of services at the edge. Prioritising edge placement leads to the lowest energy consumption, but as a consequence of the energy-latency trade-off, the task execution time may be slightly higher compared to cloud execution, potentially impacting user-perceived latency.

Table 7 presents the execution time of each algorithm. As expected, GA exhibit the longest execution times (∼4 s) due to its population-based search strategy. EMMOO follows with execution times (∼1 s) due to its iterative nature. Conversely, MROA achieves the fastest execution times across all topologies as it skips operations related to latency and energy consumption limitations when no deadlines are imposed. RTA, focusing solely on a single objective, achieves the second-fastest execution times across all topologies. EDEM and DEM demonstrate execution times comparable to RTA, making them suitable for real-time and time-sensitive applications in the MEC environment. We also conducted an additional set of experiments to assess the impact of a MEC environment. In these experiments, we assumed a setup comprising 4 MDCs, 1 Cloud DC, and a BA topology. The results show that, on average, latency is reduced by 46.8% and energy consumption by 32.9% when scheduling decisions select resources from the edge-cloud continuum instead of relying solely on cloud resources.

6.2.2. Dynamic resource scheduler evaluation

Similar to the performance evaluation of the offline placement algorithms, we conducted experiments for the four different scheduling algorithms using two distinct applications from the dataset. User requests were generated every 30 ms, and the schedulers were triggered at the same interval. The deadline for each application is pre-determined based on the method proposed in [28], with a deadline factor (γ) equals to 3.

Fig. 15 demonstrate the total number of request processing completions. The results for application jobs 3 and 10 show that the baseline scheduler, labelled as *none*, can process less than half of the user requests received due to insufficient power supply at the edge. Under the same conditions, the scaling-only scheduler occasionally fails to process some requests but still achieves a significantly higher throughput than the baseline. The offloading-only scheduler, as expected, offloads all computation jobs to the cloud and successfully processes all user requests. Meanwhile, our dynamic scaling and offloading scheduler, labelled as *DSO*, achieves the same throughput level as the offloading-only scheduler.

Fig. 16 demonstrates the average user-experienced latency of the processed requests. In both scenarios, the baseline scheduler results in extremely high latency due to the waiting time required for the batteries of the EH devices to recharge, causing task processing to be paused. The offloading-only scheduler achieves the lowest mean latency by processing all tasks on powerful machines in the cloud datacenter. This is followed by the dynamic scaling and offloading scheduler, which strikes a balance between cloud offloading and edge execution. The scaling-only scheduler performs similarly to the baseline when it fails to address the battery shortage during the processing of job 3. However, in the remaining experiments, it achieves a mean latency slightly higher than the DSO scheduler.

Fig. 17 illustrates the number of requests processed that meet the deadline. The results for the baseline scheduler reveal that without performing further actions, energy shortages not only reduce the throughput of request processing but also result in a low acceptance rate for responses. By scaling down the operating frequency of edge servers or offloading computations to the cloud, the system becomes more capable of processing requests within acceptable latency. However, the acceptance ratio for the scaling-only-EDEM case remains relatively low.

By analysing Figs. 15, 16, 17 alongside the conclusions from the offline algorithm evaluations, the difference between scaling-only-DEM and scaling-only-EDEM when processing job 3 becomes clear. DEM prioritises energy saving, enabling it to perform better under energy constraints. In contrast, EDEM focuses more on processing requests quickly, which leads to poorer performance compared to DEM when energy supply is insufficient.

Fig. 18 shows the total energy consumption resulting from the computations in the servers. Since the edge datacenters are entirely powered by EH equipment, only harvested energy is consumed by the baseline and scaling-only schedulers. The baseline scheduler, lacking any scaling policy, consumes a significant amount of edge energy while processing fewer requests. As previously explained, the scaling policy dynamically adjusts the operating frequency of edge machines during slack time. This ensures that when the response times of requests remain within an acceptable range, energy consumption is significantly reduced. Consequently, the edge energy consumed per request by the scaling-only scheduler is much lower than that of the baseline scheduler. The DSO scheduler, unlike the scaling-only policy, utilises cloud servers when necessary. As a result, it consumes additional energy by offloading part of the computations to the cloud. However, compared to the offloading-only scheduler, the energy consumption of the DSO scheduler is substantially lower.

Next, we demonstrate energy savings in terms of carbon footprint reductions and for that we study the case of Job 3 using the results obtained from policies (a) Dynamic Scaling and Offloading (DSO) and (b) Offloading-only after applying DEM and EDEM algorithms (results were averaged).

During the 20 000 timestamps simulated, our proposed strategy DSO resulted in an average energy consumption of 8.825 kWh, compared to 28.79 kWh consumed by the offloading-only approach under the same workload. This represents a grid energy saving of 19.97 kWh. To quantify the environmental benefit, we use the average US grid carbon intensity factor of 0.37 kg CO2e/kWh (based on EPA eGRID

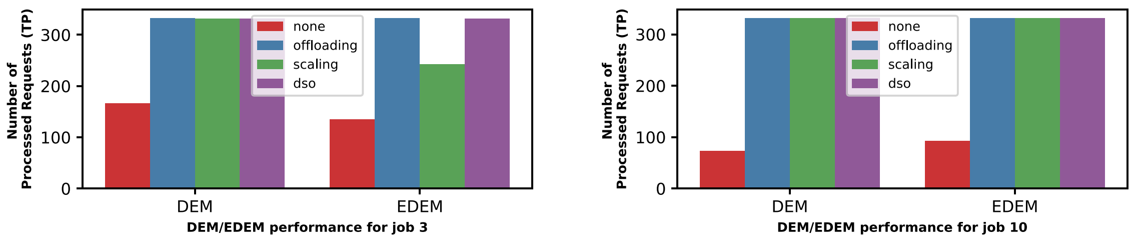


Fig. 15. Number of processed requests (TP).

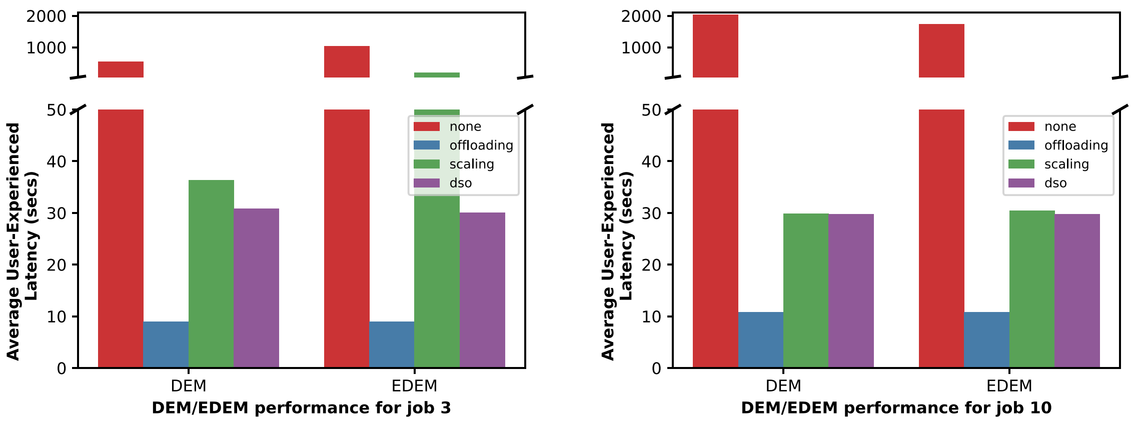


Fig. 16. Average user-experienced latency (LT).

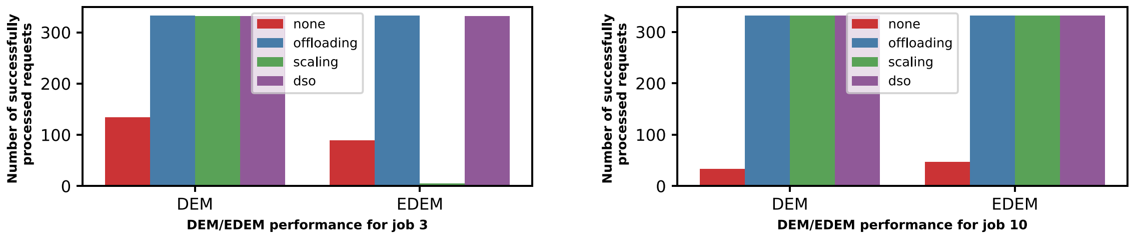


Fig. 17. The acceptance rate of processed requests (AC).

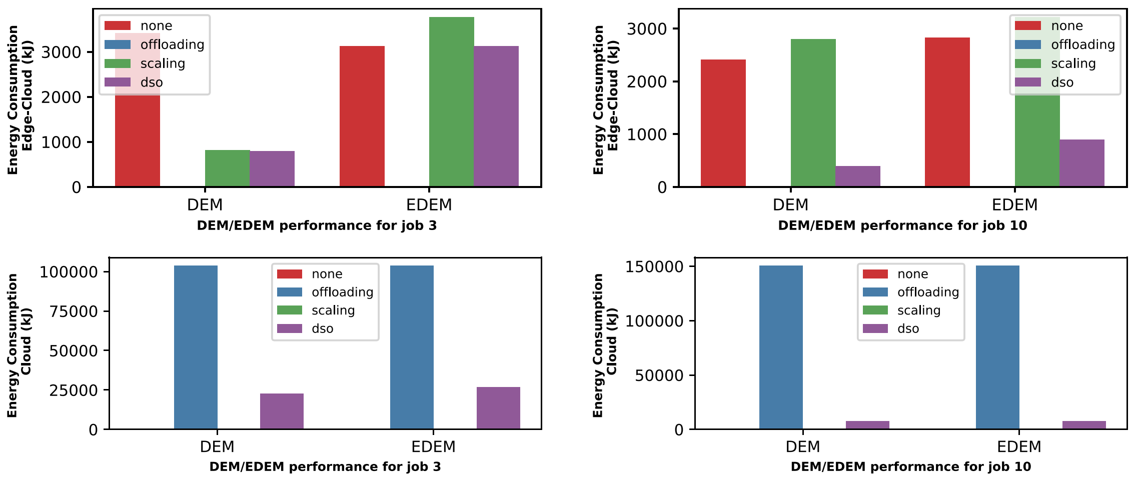


Fig. 18. Overall energy consumption (EC). The first row shows the energy consumption at the edge, while the second row shows the consumption at the cloud.

data for 2023 [32]). This translates to an estimated carbon footprint reduction of 7.39 kg CO2e achieved by our algorithm compared to the offloading-only strategy.

Fig. 19 compares the cloud rental costs of the offloading-only scheduler and the DSO scheduler. The figure shows that the DSO scheduler incurs significantly lower operational costs while maintaining strong performance in both response latency and throughput. Although its average latency is slightly higher than that of the offloading-only scheduler, it still meets the deadline constraints and ranks as the second lowest in latency.

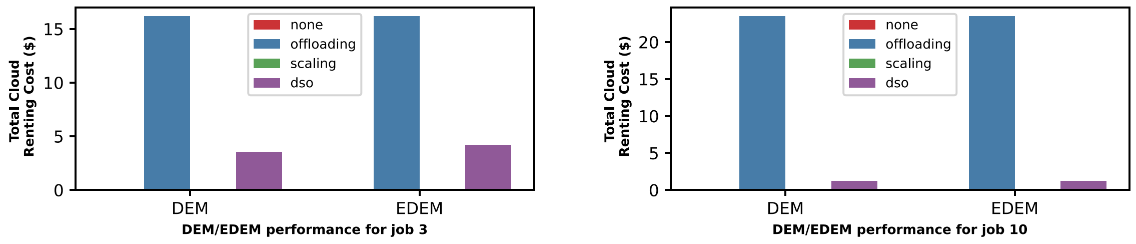


Fig. 19. Cloud Renting Cost (CRC).

In summary, these evaluations demonstrate that the DSO scheduler effectively balances energy efficiency, performance, and cost. Unlike the baseline scheduler, which suffers from low throughput and high latency due to edge energy shortages, the DSO scheduler achieves full request processing throughput while maintaining latency within the deadlines. It outperforms the scaling-only scheduler by leveraging both cloud offloading and edge execution, resulting in significantly lower energy consumption compared to the offloading-only scheduler. Furthermore, the DSO scheduler reduces cloud rental costs while delivering comparable performance in response latency and throughput. These results prove the DSO scheduler's ability to provide a sustainable and cost-effective solution for managing workloads in energy-constrained environments.

6.2.3. *Further discussion*

While our results demonstrate that the proposed algorithms outperform state-of-the-art approaches and exhibit relative stability as network complexity increases, suggesting good scalability potential, it is important to acknowledge that these evaluations were conducted under specific assumed conditions. These included relatively constant task arrival rates, typical energy harvesting availability, and average workload characteristics. However, real-world scenarios often involve extreme conditions, such as sudden traffic spikes or prolonged periods of low renewable energy availability (e.g., due to adverse weather). Considering these cases provides further insight into the robustness of our solutions.

If task arrival rates or computational demands fluctuate rapidly, our heuristic-based algorithms are designed to react quickly by generating new placement decisions upon detecting sub-deadline violations. Despite this rapid reaction capability, overall end-to-end latency could still degrade due to the potential backlog of subtasks accumulating during the time required for service module migration. This suggests that incorporating techniques such as service replication, potentially as future work, could enhance load balancing and responsiveness under such highly dynamic conditions.

Similarly, concerns arise under conditions where renewable energy sources become unavailable for extended periods and battery reserves are low. In such situations, the current algorithms would likely prioritise cloud offloading to maintain service availability where possible. While this preserves functionality, it would inevitably increase end-to-end latency and operational costs due to cloud usage. This highlights the importance of incorporating energy prediction and potentially more sophisticated energy management techniques—planned enhancements for future work—to enable our strategies to adapt more gracefully and efficiently to variable energy supply conditions.

7. Conclusions and future work

This paper firstly introduces EDEM and DEM, two heuristic-based algorithms for service module placement in MEC networks that consider dependencies between service modules. EDEM prioritises energy efficiency while maintaining low latency impact on users. DEM, on the other hand, achieves significant energy reductions by allowing for a more flexible trade-off with increased latency. Based on the two algorithms developed, we integrate the energy-harvesting technique in the MEC-Cloud environment, and designed an online resource scheduling strategy that considers energy and latency constraints and dynamically makes server frequency scaling and service offloading decisions. Simulation results showed that our scheduler effectively balances energy efficiency, cost savings, and performance, making it a more sustainable and practical approach for handling user requests in energy-constrained environments.

Our future work will incorporate user mobility models. While the current system focuses on scenarios with static users—common in many edge computing applications—extending it to support mobile users will broaden its applicability to dynamic contexts such as connected vehicles or mobile augmented reality. Developing mobility-aware scheduling algorithms can enable the system to maintain performance and service continuity as user devices roam in the network, thereby enhancing its robustness and scalability in dynamic environments.

Another key direction for future research is incorporating energy prediction models into our scheduler. The instability of energy supplies is a common challenge in real-world edge deployments relying on renewable sources like solar power. Based on sources like historical data and weather forecasts, the scheduler may proactively adjust the resource allocation plan beforehand. This would significantly enhance the applicability of our solution to energy-constrained or off-grid scenarios.

CRediT authorship contribution statement

Shuyi Chen: Software, Visualization, Writing – original draft, Methodology. **Panagiotis Oikonomou:** Writing – original draft, Methodology, Writing – review & editing, Validation, Visualization. **Zhengchang Hua:** Software, Writing – review & editing, Investigation. **Nikos Tziritas:** Methodology, Supervision, Conceptualization. **Karim Djemame:** Supervision, Investigation, Funding acquisition. **Nan Zhang:** Writing – review & editing. **Georgios Theodoropoulos:** Supervision, Project administration, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported in part by the Research Institute of Trustworthy Autonomous Systems (RITAS), and in part by the Shenzhen Science and Technology Program (project No. GJHZ20210705141807022), and by the SUSTech-University of Leeds Joint PhD Programme.

Data availability

Data will be made available on request.

References

[1] P. Oikonomou, A. Karanika, C. Anagnostopoulos, K. Kolomvatsos, On the use of intelligent models towards meeting the challenges of the edge mesh, ACM Comput. Surv. 54 (6) (2021) 1–42.

[2] F.K. Shaikh, S. Zeadally, E. Exposito, Enabling technologies for green internet of things, IEEE Syst. J. 11 (2) (2015) 983–994.

[3] F.A. Salaht, F. Desprez, A. Lebre, An overview of service placement problem in fog and edge computing, ACM Comput. Surv. 53 (3) (2020) 1–35.

[4] G. Zhang, W. Zhang, Y. Cao, D. Li, L. Wang, Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices, IEEE Trans. Ind. Inform. 14 (10) (2018) 4642–4655.

[5] Y. Zhang, J. Chen, Y. Zhou, L. Yang, B. He, Y. Yang, Dependent task offloading with energy-latency tradeoff in mobile edge computing, IET Commun. 16 (17) (2022) 1993–2001.

[6] I. Lera, C. Guerrero, C. Juiz, YAFS: A simulator for IoT scenarios in fog computing, IEEE Access 7 (2019) 91745–91758.

[7] D. Harutyunyan, N. Shahriar, R. Boutaba, R. Riggio, Latency and mobility–aware service function chain placement in 5G networks, IEEE Trans. Mob. Comput. 21 (5) (2020) 1697–1709.

[8] K. Kaur, S. Garg, G. Kaddoum, S.H. Ahmed, M. Atiquzzaman, KEIDS: Kubernetes-based energy and interference driven scheduler for industrial IoT in edge-cloud ecosystem, IEEE Internet Things J. 7 (5) (2019) 4228–4237.

[9] C. Centofanti, W. Tiberti, A. Marotta, F. Graziosi, D. Cassioli, Taming latency at the edge: A user-aware service placement approach, Comput. Netw. 247 (2024) 110444.

[10] A. Lakhan, M.A. Mohammed, O.I. Obaid, C. Chakraborty, K.H. Abdulkareem, S. Kadry, Efficient deep-reinforcement learning aware resource allocation in SDN-enabled fog paradigm, Autom. Softw. Eng. 29 (2022) 1–25.

[11] M. Adhikari, S.N. Srirama, Multi-objective accelerated particle swarm optimization with a container-based scheduling for Internet-of-Things in cloud environment, J. Netw. Comput. Appl. 137 (2019) 35–61.

[12] J. Mo, J. Liu, Z. Zhao, Exploiting function-level dependencies for task offloading in edge computing, in: IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, IEEE, 2022, pp. 1–6.

[13] L.X. Nguyen, Y.K. Tun, T.N. Dang, Y.M. Park, Z. Han, C.S. Hong, Dependency tasks offloading and communication resource allocation in collaborative UAVs networks: A meta-heuristic approach, IEEE Internet Things J. (2023).

[14] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, P. Leitner, Optimized IoT service placement in the fog, Serv. Oriented Comput. Appl. 11 (4) (2017) 427–443.

[15] A. Mebrek, L. Merghem-Boulahia, M. Esseghir, Efficient green solution for a balanced energy consumption and delay in the IoT-Fog-Cloud computing, in: 2017 IEEE 16th International Symposium on Network Computing and Applications, NCA, IEEE, 2017, pp. 1–4.

[16] X. Cai, H. Kuang, H. Hu, W. Song, J. Lü, Response time aware operator placement for complex event processing in edge computing, in: International Conference on Service-Oriented Computing, Springer, 2018, pp. 264–278.

[17] Q. Peng, Y. Xia, Y. Wang, C. Wu, X. Luo, J. Lee, Joint operator scaling and placement for distributed stream processing applications in edge computing, in: International Conference on Service-Oriented Computing, Springer, 2019, pp. 461–476.

[18] X. Zhao, Y. Shi, S. Chen, MAESP: Mobility aware edge service placement in mobile edge networks, Comput. Netw. 182 (2020) 107435.

[19] Y. Shang, J. Li, X. Wu, Dag-based task scheduling in mobile edge computing, in: 2020 7th International Conference on Information Science and Control Engineering, ICISCE, IEEE, 2020, pp. 426–431.

[20] F. Zhao, Y. Chen, Y. Zhang, Z. Liu, X. Chen, Dynamic offloading and resource scheduling for mobile-edge computing with energy harvesting devices, IEEE Trans. Netw. Serv. Manag. 18 (2) (2021) 2154–2165.

[21] H. Hu, Q. Wang, R.Q. Hu, H. Zhu, Mobility-aware offloading and resource allocation in a MEC-enabled IoT network with energy harvesting, IEEE Internet Things J. 8 (24) (2021) 17541–17556.

[22] S. Ijaz, E.U. Munir, S.G. Ahmad, M.M. Rafique, O.F. Rana, Energy-makespan optimization of workflow scheduling in fog–cloud computing, Comput. 103 (2021) 2033–2059.

[23] M. Xu, W. Li, X. Zhang, Q. Su, A discrete dwarf mongoose optimization algorithm to solve task assignment problems on smart farms, Clust. Comput. (2024) 1–20.

[24] J. Zhang, Y. Zhai, Z. Liu, Y. Wang, A Lyapunov based resource allocation method for edge-assisted industrial internet of things, IEEE Internet Things J. (2024).

[25] N. Hudson, H. Khamfroush, M. Baughman, D.E. Lucani, K. Chard, I. Foster, QoS-aware edge AI placement and scheduling with multiple implementations in FaaS-based edge computing, Future Gener. Comput. Syst. 157 (2024) 250–263.

[26] Z. Niu, H. Liu, J. Du, Y. Ge, Partial task offloading for UAV-assisted mobile edge computing with energy harvesting, IEEE Internet Things J. (2024).

[27] S. Chen, P. Oikonomou, Z. Hua, N. Tziritas, K. Djemame, N. Zhang, G. Theodoropoulos, Efficient placement of interdependent services in multi-access edge computing, in: 20th Conference on the Economics of Grids, Clouds, Software, and Services, Springer, 2024.

[28] X. Ma, H. Xu, H. Gao, M. Bian, Real-time multiple-workflow scheduling in cloud environments, IEEE Trans. Netw. Serv. Manag. 18 (4) (2021) 4002–4018.

[29] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Sci. 286 (5439) (1999) 509–512.

[30] D.J. Watts, S.H. Strogatz, Collective dynamics of 'small-world' networks, Nat. 393 (6684) (1998) 440–442.

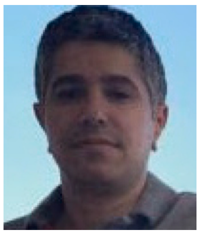
[31] N. Sarrafzade, R. Entezari-Maleki, L. Sousa, A genetic-based approach for service placement in fog computing, J. Supercomput. 78 (8) (2022) 10854–10875.

[32] United States Environmental Protection Agency (EPA), Emissions & generation resource integrated database (eGRID), 2023, https://www.epa.gov/egrid. (Accessed 14 April 2025).

Shuyi Chen is a Ph.D. candidate in the SUSTech-University of Leeds Joint Ph.D. programme. Her research interests include resource management, distributed systems, and multi-access edge computing. She holds a BEng from the Southern University of Science and Technology (SUSTech), China.

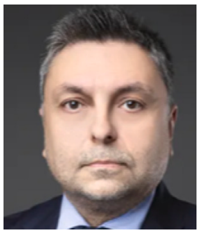
Panagiotis Oikonomou received the diploma and M.Sc. degrees from the Department of Electrical and Computer Engineering, University of Thessaly, Greece, in 2008 and 2010, respectively, and the Ph.D. degree in Temporospatial organisation of circuits and tasks over the Cloud from the Department of Electrical and Computer Engineering, University of Thessaly. He is currently a scientific scholarship holder and a postdoc researcher in scheduling and orchestration of tasks in the cloud. His research interests include CAD algorithms, optimisation algorithms, and cloud computing.

Zhengchang Hua is a Ph.D. candidate in the SUSTech-University of Leeds Joint Ph.D. programme. His research interests include digital twin networks, agent-based systems, and discrete event simulation. He holds a BEng from the Southern University of Science and Technology, China.


Nikos Tziritas is an Associate Professor with the Department of Informatics and Telecommunications at the University of Thessaly, Greece, where he also obtained his PhD. Prior to this he was an Associate Professor at the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, China. He is the recipient of the 2016 Award for Excellence for Early Career Researchers in Scalable Computing from IEEE Technical Committee in Scalable Computing.

Karim Djemame is a Professor of distributed systems in the School of Computing at the University of Leeds, UK. His research interests are distributed systems; cloud computing; energy efficiency; heterogeneous parallel architectures; performance evaluation. He holds a Ph.D. from the University of Glasgow, UK.

Nan Zhang is a postdoctoral researcher in the Department of Computer Science and Engineering at Southern University of Science and Technology, China. His research interests include network digital twins, distributed systems, self-aware and self-adaptive systems, and parallel and distributed simulation. He holds a Ph.D. from the University of Birmingham, UK, and a BEng from SUSTech, China.



Georgios Theodoropoulos is currently a Chair Professor at the Department of Computer Science and Engineering at SUSTech in Shenzhen. In the past he has held senior appointments at Universities of Durham and Birmingham in the UK, IBM Research, Ireland, and Nanyang Technological University, Singapore. He holds a Ph.D. from the University of Manchester, UK. He is a Member of the European Academy of Sciences and Arts and a Fellow of the World Academy of Art and Science.